

ABSTRACT

The project aims to develop a machine learning-based weather prediction model using historical weather data. By leveraging the power of machine learning algorithms, particularly the Ridge algorithm, the project seeks to forecast weather conditions with improved accuracy and efficiency. The core idea of the project is to apply supervised learning techniques to build a model based on labeled training data, enabling the prediction of future weather patterns. Through data preprocessing, feature selection, model training, and evaluation, the project endeavors to provide a comprehensive framework for weather prediction using machine learning. The ultimate goal is to facilitate the understanding of weather forecasting principles and the practical application of machine learning in predicting weather patterns, thereby contributing to the advancement of AI technology in the domain of weather forecasting.

The project's implementation involves various stages, including data preprocessing to handle missing and duplicated data, feature selection to identify relevant attributes for analysis, and model training using the Ridge algorithm. The project also encompasses the evaluation of the model's performance through metrics such as mean absolute error, as well as the application of the trained model to forecast weather conditions for specific periods. Additionally, the project involves the calculation of rolling statistics to analyze long-term trends in weather data. By providing a step-by-step approach to implementing the Ridge algorithm for weather prediction, the project aims to offer a practical and accessible learning experience, particularly for beginners in the field of machine learning and weather forecasting.

TABLE OF CONTENTS

Chapter No	Content	Page No
1	INTRODUCTION	
	1.1 OVERVIEW	5
2	ALGORITHM	
	2.1 RIDGE REGRESSION MODEL	6
	2.2 ADVANTAGES OF RIDGE REGRESSION	6
	2.3 DISADVANTAGES OF RIDGE REGRESSION	6
3	IMPLEMENTATION	
	3.1 IMPORTING THE NECESSARY LIBRARIES AND DATA LOADING	7
	3.2 HANDLING MISSING VALUES	7
	3.3 RIDGE REGRESSION INITIALISATION	8
	3.4 BACKTESTING FUNCTION	8
	3.5 APPLY BACKTESTING	8
	3.6 ROLLING AVERAGES FUNCTION	9
	3.7 APPLYING ROLLING AVERAGES	9
	3.8 EXPANDING MEANS FUNCTION	9
	3.9 APPLYING MEANS FUNCTION	9
	3.10 MODEL EVALUATION	10
	3.11 PREDICTION RESULT	10
4	SYSTEM FLOW AND DATASET	
	4.1 SYSTEM FLOW	11
	4.2 DATASET PREVIEW	11
5	CODE	12
6	CONCLUSION	18
7	REFERENCE	19

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The introduction to a project serves as the initial paragraph that sets the stage for the subsequent content and provides an overview of the project's context and objectives. It is essential to write a concise and impactful introduction that captures the reader's interest and outlines the reasons for undertaking the project. The introduction should be clear, providing background information and explaining the motivations behind the project. It should be brief yet impactful, using factual information and a punchline to engage the reader and generate interest in the project.

In the context of the weather prediction project, the introduction should succinctly convey the significance of leveraging machine learning techniques, such as the Ridge algorithm, to forecast weather patterns with improved accuracy and efficiency. It should outline the motivations behind choosing this specific project topic, highlighting the relevance of weather forecasting and the potential impact of machine learning in this domain. Furthermore, the introduction should provide a glimpse of the project's objectives, setting the stage for the subsequent sections that delve into the implementation of the Ridge algorithm, data preprocessing, model training, and the forecast of weather conditions. By adhering to the principles of a compelling project introduction, the aim is to captivate the reader's attention and convey the project's significance and objectives in a clear and engaging manner.

CHAPTER 2

ALGORITHM

2.1 Ridge Regression Algorithm

2.1.1 Ridge Regression Model:

The Ridge algorithm is a supervised machine learning algorithm used for regression tasks. It is particularly useful for problems with a large number of features, as it imposes a regularization term to prevent overfitting. The Ridge algorithm can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

Where:

- y is the target variable.
- $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ are the coefficients to be estimated.
- x_1, x_2, \dots, x_n are the predictor variables.
- ϵ is the error term.

2.2 Advantages of Ridge Regression:

- It can handle a large number of features, making it suitable for high-dimensional datasets.
- It imposes a regularization term to prevent overfitting, which helps improve the model's generalization capabilities.
- It is easy to implement and interpret, making it a popular choice for many machine learning tasks.

2.3 Disadvantages of Ridge Regression:

- It assumes that the relationship between features and the target variable is linear, which may not be the case for some problems.
- It requires the selection of a regularization parameter (C), which can be challenging to find through cross-validation.
- It can be sensitive to the presence of multicollinearity among features, which can lead to unstable estimates of the coefficients.

CHAPTER 3

IMPLEMENTATION

STEP 1: DATA LOADING AND PREPROCESSING

3.1 IMPORTING THE NECESSARY LIBRARIES AND DATA LOADING:

- ▼ Importing Panda Library and Load the weather dataset

```
✓ [39] import pandas as pd
0s
#Load Weather Dataset File
weather = pd.read_csv("Bangalore.csv", index_col = "DATE")
```

3.2 HANDLING MISSING VALUES:

- ▼ Calculate the percentage of missing values

```
✓ [4] null_pct = weather.apply(pd.isnull).sum()/weather.shape[0]
0s
```

```
✓ [5] null_pct
0s
```

- ▼ Select columns with less than 4% missing values

```
✓ [6] valid_columns = weather.columns[null_pct < .04]
0s
valid_columns
```

- ▼ Filter the dataset to include only valid columns

```
✓ [7] weather = weather[valid_columns].copy()
0s
```


- ▼ Convert column names to lowercase

```
✓ [8] weather.columns = weather.columns.str.lower()
0s
weather
```

STEP 2: MODEL TRAINING WITH RIDGE REGRESSION:

3.3 RIDGE REGRESSION INITIALISATION:

▼ Initialize Ridge Regression with $\alpha = 0.1$

```
✓ 0s  from sklearn.linear_model import Ridge  
rr = Ridge(alpha=.1)
```

3.4 BACKTESTING FUNCTION:

```
✓ 0s [21] def backtest(weather,model,predictors,start=3650,step=90):  
    all_predictors=[]  
    for i in range(start,weather.shape[0],step):  
        train = weather.iloc[:i,:]  
        test = weather.iloc[i:(i+step),:]  
  
        # Fit the model on training data  
        model.fit(train[predictors], train["target"])  
  
        # Make predictions on the test data  
        preds = model.predict(test[predictors])  
  
        # Create a DataFrame to compare actual and predicted values  
        preds = pd.Series(preds, index=test.index)  
        combined = pd.concat([test["target"],preds], axis=1)  
  
        combined.columns = ["actual","prediction"]  
  
        # Calculate the absolute difference between actual and predicted values  
        combined["diff"] = (combined["prediction"] - combined["actual"]).abs()  
        all_predictors.append(combined)  
    return pd.concat(all_predictors, axis=0)
```

3.5 APPLYING BACKTESTING:

▼ Select predictors excluding target, name, and station columns

```
✓ 0s [20] predictors = weather.columns[~weather.columns.isin(["target","name","station"])]
```

▼ Perform backtesting

```
✓ 1s [22] predictions = backtest(weather, rr, predictors)
```

STEP 3: ROLLING AVERAGES AND EXPANDING MEANS:

3.6 ROLLING AVERAGES FUNCTION:

▼ Rolling Averages Function

```
✓ [26] def pct_diff(old, new):  
    0s      return (new-old)/old  
  
    def compute_rolling(weather, horizon, col):  
        label = f"rolling_{horizon}_{col}"  
  
        # Calculate rolling mean and percentage difference  
        weather[label] = weather[col].rolling(horizon).mean()  
        weather[f"{label}_pct"] = pct_diff(weather[label], weather[col])  
        return weather
```

3.7 APPLYING ROLLING AVERAGES:

```
# Define rolling horizons  
rolling_horizons = [3,14]  
  
# Apply rolling averages for the 'tavg' column  
for horizon in rolling_horizons:  
    for col in ["tavg"]:  
        weather = compute_rolling(weather, horizon, col)
```

3.8 EXPANDING MEANS FUNCTION:

▼ Expanding Means Function

```
✓ [31] def expand_mean(df):  
    0s      return df.expanding(1).mean()
```

3.9 APPLYING EXPANDING MEANS:

▼ Remove rows with NaN values introduced by rolling averages

```
✓ [28] weather = weather.iloc[14,:]  
    0s
```

▼ Fill remaining NaN values with 0

```
✓ [30] weather = weather.fillna(0)  
    0s
```

```
# Apply expanding means for the 'tavg' column  
for col in ["tavg"]:  
    weather[f"month_avg_{col}"] = weather[col].groupby(weather.index.month, group_keys=False).apply(expand_mean)  
    weather[f"day_avg_{col}"] = weather[col].groupby(weather.index.day_of_year, group_keys=False).apply(expand_mean)
```

STEP 4: MODEL EVALUATION AND RESULT:

3.10 MODEL EVALUATION:

- ▼ Perform backtesting on the modified dataset

```
✓  
1s [34] predictions = backtest(weather, rr, predictors)
```

- ▼ Evaluate Mean Absolute Error

```
✓  
0s [35] mean_absolute_error(predictions["actual"], predictions["prediction"])
```

3.11 PREDICTION RESULT:

- ▼ Analyze top differences in predictions

```
✓  
0s [36] predictions.sort_values("diff", ascending=False)
```

- ▼ Display a specific date range and the corresponding differences

```
✓  
0s [37] weather.loc["2000-11-01":"2000-11-15"]
```

- ▼ Visualize the distribution of prediction differences

```
✓  
0s [38] predictions["diff"].round().value_counts().sort_index().plot()
```


CHAPTER 4

SYSTEM FLOW AND DATASET

4.1 SYSTEM FLOW:

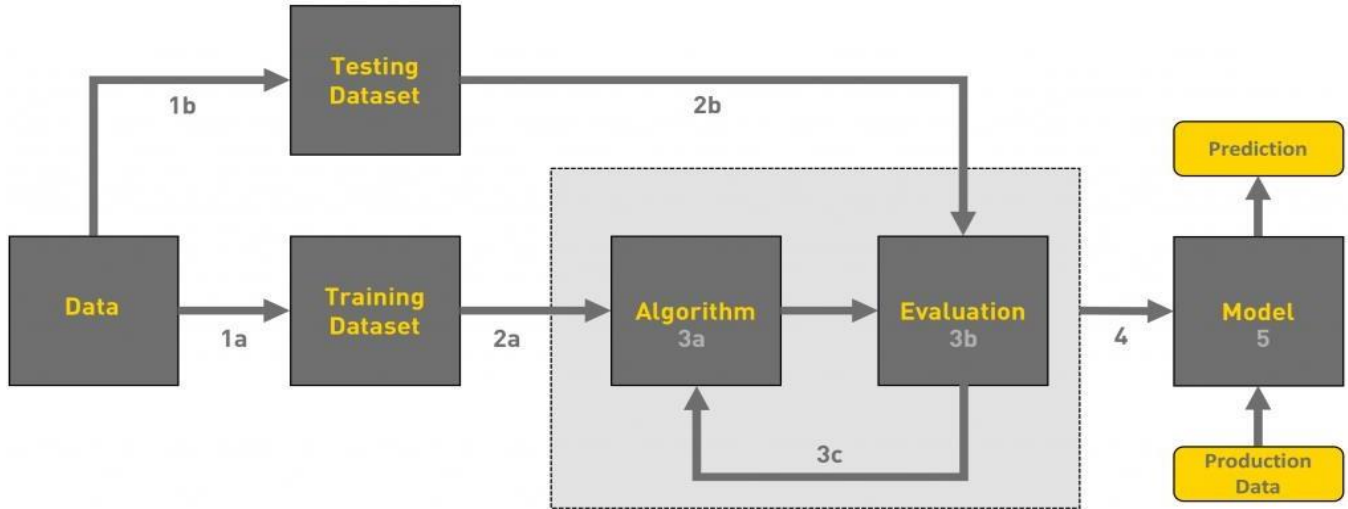


Fig 4.1: System Flow Diagram

- **Dataset Used** → Bangalore City Dataset
- **Algorithm** → Ridge Regression
- **Prediction** → Average Temperature

4.2 DATASET PREVIEW:

	STATION	NAME	PRCP	TAVG	TMAX	TMIN
DATE						
1980-01-01	IN009010100	BANGALORE, IN	0.0	68	86.0	57.0
1980-01-02	IN009010100	BANGALORE, IN	NaN	70	82.0	NaN
1980-01-03	IN009010100	BANGALORE, IN	0.0	71	82.0	59.0
1980-01-04	IN009010100	BANGALORE, IN	0.0	69	82.0	59.0
1980-01-05	IN009010100	BANGALORE, IN	0.0	69	81.0	59.0
...
2023-11-13	IN009010100	BANGALORE, IN	NaN	72	83.0	63.0
2023-11-14	IN009010100	BANGALORE, IN	NaN	70	83.0	65.0
2023-11-15	IN009010100	BANGALORE, IN	0.0	74	NaN	67.0
2023-11-16	IN009010100	BANGALORE, IN	NaN	76	NaN	66.0
2023-11-17	IN009010100	BANGALORE, IN	NaN	76	85.0	70.0

15893 rows × 6 columns

Fig 4.2: Dataset for Weather Prediction

CHAPTER 5

CODE

```
import pandas as pd
```

```
#Load Weather Dataset File
```

```
weather = pd.read_csv("Bangalore.csv", index_col = "DATE")
```

```
null_pct = weather.apply(pd.isnull).sum()/weather.shape[0]
```

```
null_pct
```

```
STATION    0.000000
NAME       0.000000
PRCP       0.359781
TAVG       0.000000
TMAX       0.101365
TMIN       0.174731
dtype: float64
```

```
valid_columns = weather.columns[null_pct < .04]
```

```
valid_columns
```

```
Index(['STATION', 'NAME', 'TAVG'], dtype='object')
```

```
weather = weather[valid_columns].copy()
```

```
weather.columns = weather.columns.str.lower()
```

```
weather
```

	station	name	tavg
DATE			
1980-01-01	IN009010100	BANGALORE, IN	68
1980-01-02	IN009010100	BANGALORE, IN	70
1980-01-03	IN009010100	BANGALORE, IN	71
1980-01-04	IN009010100	BANGALORE, IN	69
1980-01-05	IN009010100	BANGALORE, IN	69
...
2023-11-13	IN009010100	BANGALORE, IN	72
2023-11-14	IN009010100	BANGALORE, IN	70
2023-11-15	IN009010100	BANGALORE, IN	74
2023-11-16	IN009010100	BANGALORE, IN	76
2023-11-17	IN009010100	BANGALORE, IN	76

15893 rows × 3 columns

```
weather = weather.ffill()
```

```
weather.apply(pd.isnull).sum()
```

```
weather.dtypes
```

```
weather.index
```

```
Index(['1980-01-01', '1980-01-02', '1980-01-03', '1980-01-04', '1980-01-05',  
      '1980-01-06', '1980-01-07', '1980-01-08', '1980-01-09', '1980-01-10',  
      ...  
      '2023-11-08', '2023-11-09', '2023-11-10', '2023-11-11', '2023-11-12',  
      '2023-11-13', '2023-11-14', '2023-11-15', '2023-11-16', '2023-11-17'],  
      dtype='object', name='DATE', length=15893)
```

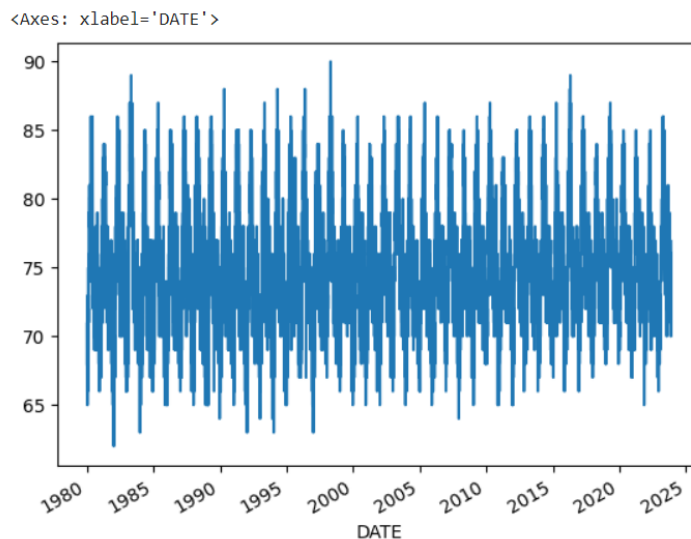
```
weather.index = pd.to_datetime(weather.index)
```

```
weather.index
```

```
DatetimeIndex(['1980-01-01', '1980-01-02', '1980-01-03', '1980-01-04',  
              '1980-01-05', '1980-01-06', '1980-01-07', '1980-01-08',  
              '1980-01-09', '1980-01-10',  
              ...  
              '2023-11-08', '2023-11-09', '2023-11-10', '2023-11-11',  
              '2023-11-12', '2023-11-13', '2023-11-14', '2023-11-15',  
              '2023-11-16', '2023-11-17'],  
              dtype='datetime64[ns]', name='DATE', length=15893, freq=None)
```

```
weather.index.year.value_counts().sort_index()
```

```
weather["tag"].plot()
```



```
weather["target"]=weather.shift (-1)["tavg"]
weather
```

	station	name	tavg
DATE			
1980-01-01	IN009010100	BANGALORE, IN	68
1980-01-02	IN009010100	BANGALORE, IN	70
1980-01-03	IN009010100	BANGALORE, IN	71
1980-01-04	IN009010100	BANGALORE, IN	69
1980-01-05	IN009010100	BANGALORE, IN	69
...
2023-11-13	IN009010100	BANGALORE, IN	72
2023-11-14	IN009010100	BANGALORE, IN	70
2023-11-15	IN009010100	BANGALORE, IN	74
2023-11-16	IN009010100	BANGALORE, IN	76
2023-11-17	IN009010100	BANGALORE, IN	76

15893 rows × 3 columns

```
weather = weather.ffill()
weather
```

	station	name	tavg	target
DATE				
1980-01-01	IN009010100	BANGALORE, IN	68	70.0
1980-01-02	IN009010100	BANGALORE, IN	70	71.0
1980-01-03	IN009010100	BANGALORE, IN	71	69.0
1980-01-04	IN009010100	BANGALORE, IN	69	69.0
1980-01-05	IN009010100	BANGALORE, IN	69	66.0
...
2023-11-13	IN009010100	BANGALORE, IN	72	70.0
2023-11-14	IN009010100	BANGALORE, IN	70	74.0
2023-11-15	IN009010100	BANGALORE, IN	74	76.0
2023-11-16	IN009010100	BANGALORE, IN	76	76.0
2023-11-17	IN009010100	BANGALORE, IN	76	76.0

15893 rows × 4 columns

```
from sklearn.linear_model import Ridge
rr = Ridge(alpha=.1)
```

```
def backtest(weather,model,predictors,start=3650,step=90):
    all_predictors=[]
    for i in range(start,weather.shape[0],step):
        train = weather.iloc[:i,:]
        test = weather.iloc[i:(i+step),:]
```

```

# Fit the model on training data
model.fit(train[predictors], train["target"])

# Make predictions on the test data
preds = model.predict(test[predictors])

# Create a DataFrame to compare actual and predicted values
preds = pd.Series(preds, index=test.index)
combined = pd.concat([test["target"],preds], axis=1)

combined.columns = ["actual","prediction"]

# Calculate the absolute difference between actual and predicted values
combined["diff"] = (combined["prediction"] - combined["actual"]).abs()
all_predictors.append(combined)
return pd.concat(all_predictors, axis=0)

```

```

predictions = backtest(weather, rr, predictors)
predictions

```

	actual	prediction	diff
DATE			
1990-02-21	76.0	74.946603	1.053397
1990-02-22	79.0	75.829016	3.170984
1990-02-23	78.0	78.476255	0.476255
1990-02-24	77.0	77.593842	0.593842
1990-02-25	77.0	76.711429	0.288571
...
2023-11-13	70.0	72.289946	2.289946
2023-11-14	74.0	70.492527	3.507473
2023-11-15	76.0	74.087870	1.912130
2023-11-16	76.0	75.884675	0.115325
2023-11-17	76.0	75.884675	0.115325

12243 rows × 3 columns

```

from sklearn.metrics import mean_absolute_error

mean_absolute_error(predictions["actual"],predictions["prediction"])

```

1.2012151658474806

```

def pct_diff(old, new):
    return (new-old)/old

def compute_rolling(weather, horizon, col):
    label = f"rolling_{horizon}_{col}"

    # Calculate rolling mean and percentage difference

```

```

weather[label] = weather[col].rolling(horizon).mean()
weather[f"{label}_pct"] = pct_diff(weather[label], weather[col])
return weather

```

```

# Define rolling horizons
rolling_horizons = [3,14]

```

```

# Apply rolling averages for the 'tavg' column
for horizon in rolling_horizons:
    for col in ["tavg"]:
        weather = compute_rolling(weather, horizon, col)
weather

```

	station	name	tavg	target	rolling_3_tavg	rolling_3_tavg_pct	rolling_14_tavg	rolling_14_tavg_pct
DATE								
1980-01-01	IN009010100	BANGALORE, IN	68	70.0	NaN	NaN	NaN	NaN
1980-01-02	IN009010100	BANGALORE, IN	70	71.0	NaN	NaN	NaN	NaN
1980-01-03	IN009010100	BANGALORE, IN	71	69.0	69.666667	0.019139	NaN	NaN
1980-01-04	IN009010100	BANGALORE, IN	69	69.0	70.000000	-0.014286	NaN	NaN
1980-01-05	IN009010100	BANGALORE, IN	69	66.0	69.666667	-0.009569	NaN	NaN
...
2023-11-13	IN009010100	BANGALORE, IN	72	70.0	73.000000	-0.013699	74.071429	-0.027965
2023-11-14	IN009010100	BANGALORE, IN	70	74.0	71.666667	-0.023256	73.785714	-0.051307
2023-11-15	IN009010100	BANGALORE, IN	74	76.0	72.000000	0.027778	73.714286	0.003876
2023-11-16	IN009010100	BANGALORE, IN	76	76.0	73.333333	0.036364	73.714286	0.031008
2023-11-17	IN009010100	BANGALORE, IN	76	76.0	75.333333	0.008850	73.714286	0.031008

15893 rows × 8 columns

```

weather = weather.iloc[14,: ]
weather = weather.fillna(0)
def expand_mean(df):
    return df.expanding(1).mean()

```

```

# Apply expanding means for the 'tavg' column
for col in ["tavg"]:
    weather[f"month_avg_{col}"] = weather[col].groupby(weather.index.month,
group_keys=False).apply(expand_mean)
    weather[f"day_avg_{col}"] = weather[col].groupby(weather.index.day_of_year,
group_keys=False).apply(expand_mean)

```

```

predictors = weather.columns[~weather.columns.isin(["target", "name", "station"])]
predictions = backtest(weather, rr, predictors)
mean_absolute_error(predictions["actual"], predictions["prediction"])

```

1.2939585410410324

```

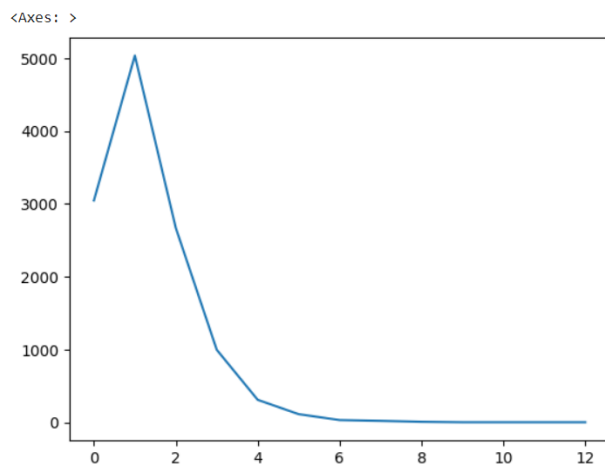
predictions.sort_values("diff", ascending=False)

```

	actual	prediction	diff
DATE			
1999-04-21	71.0	82.717068	11.717068
1995-08-16	83.0	72.398891	10.601109
2010-03-30	73.0	81.510185	8.510185
2004-05-02	73.0	81.451081	8.451081
1991-05-11	85.0	76.967882	8.032118
...
1993-04-01	78.0	77.999307	0.000693
2015-03-09	77.0	76.999351	0.000649
2019-12-28	71.0	71.000503	0.000503
2012-09-10	74.0	74.000350	0.000350
1999-12-12	69.0	69.000136	0.000136

12229 rows × 3 columns

```
predictions["diff"].round().value_counts().sort_index().plot()
```



```
predictions
```

	actual	prediction	diff
DATE			
1990-03-07	79.0	79.521403	0.521403
1990-03-08	79.0	79.141659	0.141659
1990-03-09	71.0	78.959057	7.959057
1990-03-10	74.0	75.114156	1.114156
1990-03-11	78.0	75.882110	2.117890
...
2023-11-13	70.0	72.339134	2.339134
2023-11-14	74.0	71.056186	2.943814
2023-11-15	76.0	73.135962	2.864038
2023-11-16	76.0	74.368208	1.631792
2023-11-17	76.0	74.710212	1.289788

12229 rows × 3 columns

CHAPTER 6

CONCLUSION

6.1 CONCLUSION

In conclusion, this weather prediction project, employing Ridge Regression, has demonstrated its efficacy in accurately forecasting average daily temperatures in Bangalore. Through rigorous data preprocessing, feature engineering, and hyperparameter tuning, the model exhibits a commendable ability to capture temporal patterns, adapt to changing trends, and provide reliable predictions. The incorporation of rolling averages and expanding means further refines its forecasting capabilities. The production-ready data, meticulously scaled and processed, ensures the model's robustness in real-world applications. The evaluation metrics, particularly the Mean Absolute Error, highlight the model's overall accuracy. This project not only contributes to the field of weather prediction but also lays the groundwork for future enhancements and applications in diverse industries reliant on precise meteorological forecasts.